An attempt to use SNNs instead of ANNs to solve differential equations

Hannah Köster[1], T. Schönfeldt[2], Wookyung Lee[3], Boon Tao Chew[4], and Philipp Bauer[5]

[1]hakoester@uni-osnabrueck.de
[4]bchew@uni-osnabrueck.de
[5]phibauer@uni-osnabrueck.de

Neurodynamics, SoSe 2022 *
24th July 2022

1 Introduction

Neural networks (NNs) have been shown (Hornik, 1991) to be able to approximate functions. In particular, Physics-Informed Neural Networks (PINNs) are used to efficiently solve partial differential equations (PDEs) non-numerically (Kharazmi et al., 2019). Although Artificial Neural Networks (ANNs) have enabled groundbreaking advances, they rely on biologically inaccurate neuron models, and training is associated with high energy consumption (Kundu et al., 2021). Spiking Neural Networks (SNNs) promise to solve these issues, because they use more biologically plausible models of neurons (Li & Furber, 2021). SNNs utilize spikes, which are discrete events that occur at distinct time points, as opposed to continuous values.

Theoretically, SNNs should also be able to solve differential equations (DEs) according to similar principles as used by ANNs (Tavanaei et al., 2019), since they can be converted into each other. Nevertheless, the ability to solve DEs with SNNs has rarely been investigated. Therefore, the research objective is to investigate whether and to what extent the said DEs can be solved with SNNs instead of ANNs. First, we will train ANNs using TensorFlow, and convert them into SNNs with SNN toolbox (Rueckauer et al., 2017). After the conversion, we planned to compare the differences of root-mean-square deviation (RMSD) between ANNs and SNNs. Additionally, we will compare performance of ANN when used to solve DEs of different order and different linearity.

* Word Count: 2719 words

With the objective in mind, we hypothesized the following by comparing ANN and rate-coded SNN implementations of PINNs:

1. Linear, lower order DEs are solved more accurately (less final losses) and with less training time than nonlinear higher order DEs.
2. SNNs are not more accurate than ANNs, i.e. having greater RMSD in solving DEs.

1.1 Individual Roles

Table 1: Individual roles.

Group Member	**Roles**
Hannah Köster	Implementation and training of ANNs
	Conversion of ANNs to SNNs
	Plotting the results
	Documentation on code implementation
	Proofreading of poster and short paper
Thore Schönfeldt	Training of ANNs
	Literature review
	Proofreading of poster and short paper
Wookyung Lee	Project management
	Provision of list of differential equations for code implementation
	Outcome analysis
	Composition of poster and short paper
Boon Tao Chew	Provision of list of differential equations for code implementation
	Outcome analysis
	Composition of poster and short paper
Philipp Bauer	Literature review
	Access to SpiNNaker

Continued on next page

Table 1: Individual roles. (Continued)

Group Member	Roles
	Composition of poster and short paper

2 Literature Review

2.1 Physics-Informed Neural Networks (PINNs)

Solving PDEs plays a crucial role in modeling natural dynamical systems, as done e.g. in physics, engineering, earth sciences or biophysics. Solving PDEs computationally has recently become a busy field of research within the discipline of Scientific Machine Learning. PINNs allow to integrate real-world data and mathematical physics models (in the form of DEs). This is even possible when the physical process is only partially understood, data is noisy, or in high-dimensional spaces. PINNs allow to not just fit any function to the data, but a function that obeys the constraints set out by the DEs, which express some known physical properties of the underlying process (Karniadakis et al., 2021). The general idea of PINNs is to include the mathematical formula of the DE in the loss calculation, which leads to an approximation of a solution for the DE. This ensures that the results presented by the network will comply with the known physical constraints and prevents potential contradictions already at the training stage. In principle, this also works without any data, only using the formula for the loss calculation (Cuomo et al., 2022).

2.2 Spiking Neural Networks (SNNs)

SNNs are networks of interconnected spiking neuron models. Compared to ANNs, they rely on discrete spikes for information transmission. This facilitates sparser information encoding, and as a consequence more computationally and energy efficient algorithms. Yet, SNNs are not optimally suited for classical Von-Neumann architectures. To leverage their full potential, vast parallelization of computations is needed, which has led to ongoing research in neuromorphic hardware (Javanshir et al., 2022).

2.2.1 Spiking Neuron Models

Artificial neurons use activation functions such as sigmoid, ReLu, etc. and have no memory, whereas spiking neuron models use a non-differentiable neuron model, most often a system of DEs, and have

memory. The most common spiking neuron models are the Hodgkin-Huxley, Izhikevich, (leaky) integrate-and-fire, and spike response models. The models differ in their mathematical complexity and their ability to simulate the behaviour of various biological neurons, which implies a trade-off between biological plausibility and computational efficiency (Paugam-Moisy & Bohte, 2012).

2.2.2 Information Coding

The encoding of information in a sequence of spikes is called an encoding scheme, and has significant influence on the performance of an SNN. Encoding schemes are categorized into two major classes: rate coding and temporal coding. Rate coding encodes the transmitted information in the spiking rates. In contrast, temporal coding relies on the timing of neural spikes, which makes it readily suitable for time-series processing tasks. It encompasses approaches such as latency coding, rank-order coding, phase coding and population coding. While rate-codes are easily convertible from ANNs and robust against noise, the majority of advantages lies in the field of temporal codes. Those offer sparser encodings, faster reaction times, higher transmission speed, and they are better suited to the implementation of local learning rules, such as spike-timing-dependent plasticity (STDP) (Javanshir et al., 2022).

2.2.3 Algorithms for SNNs

Unsupervised learning in SNNs relies on the STDP algorithm, which has been derived from neuroscientific research: synapses that are likely to have contributed to the neuron's firing should be reinforced, and those that did not contribute should be weakened. For supervised learning in SNNs, STDP algorithms and approaches adapting backpropagation for SNNs have been developed.

Another technique consists in pre-training an ANN, which is then converted to an SNN. This combines the advantages of using established training algorithms for ANNs and efficient neuromorphic computing. On the downside, this approach fails to make use of the neuromorphic hardware to accelerate the learning process.

Evolutionary algorithms can be used to optimize the network's topology, model hyperparameters, or synaptic weights and delays. Algorithms such as differential evolution, grammatical evolution, harmony search algorithm and particle swarm optimization have been applied to learning synaptic weights (Javanshir et al., 2022).

2.2.4 Performance Comparisons of ANN vs. SNN, and their Criticism

Although deep learning in SNNs still has not attained the same levels of accuracy compared to ANNs, the gap is decreasing, and almost equal results have been achieved on some benchmark tasks, such as the MNIST (handwritten digit recognition) benchmark. In the meanwhile, SNNs typically require much fewer operations (Tavanaei et al., 2019).

These comparisons have also been criticized (Deng et al., 2020), because SNNs may be much better suited for dynamic and temporal tasks, such as interacting with an environment or object recognition in moving images, than for the typical ANN benchmark tasks, such as static image recognition. Simply converting ANN benchmark tasks to sequences of spikes might not the best way to create a data set for neuromorphic learning.

2.2.5 Current Areas of Research

SNNs are still a young field of research. Although considerable progress has been made in the areas of neuron models, encoding schemes, learning algorithms, software and hardware implementations, challenges in all those field remain and further breakthroughs are to be expected. In particular, further research is needed in the relations of neural codings and learning to leverage efficient algorithms (Javanshir et al., 2022). On the hardware side, neuromorphic sensors that directly encode perceptions in spikes are currently being developed (Han et al., 2022).

3 Methods

3.1 Conversion/Adaptation from ANNs to SNNs

To implement the PINN as an ANN, we used the open-source library, TensorFlow. TensorFlow is not intended to support SNNs. Therefore we used the SNN toolbox library (Rueckauer et al., 2017) to convert the trained ANN to an SNN.

3.2 Differential Equations (DEs)

We based our experiments on ordinary differential equations (ODEs) and categorized them by order and linearity - *linear first order*, *linear second order*, *linear third order*, *nonlinear first order*, *nonlinear second order*, and *nonlinear third order*.

These are examples of ODEs we used in each category.

Table 2: Examples of categorized ODEs.

Category of DEs	Examples
Linear First Order	$\frac{dy}{dx} + 2xy = 0$
Linear Second Order	$\frac{d^2y}{dx^2} + y = 0$
Linear Third Order	$\frac{d^3y}{dx^3} + \frac{d^2y}{dx^2} - 6\frac{dy}{dx} + 4y = 0$
Nonlinear First Order	$\frac{dy}{dx} = 0.07y(1-\frac{y}{900})$
Nonlinear Second Order	$\frac{d^2y}{dx^2} + 3y^2\frac{dy}{dx} = 0$
Nonlinear Third Order	$\frac{d^3y}{dx^3} + (\frac{dy}{dx})^2 - y\frac{d^2y}{dx^2} = 0$

3.3 Measures for Performance Comparison

We took the final losses and the training time as comparison criteria for comparing performances between the ANNs which solve different DEs. For comparing the ANNs' and SNNs' performance, we planned to compare the RMSD of the ANNs and the SNNs.

3.4 Documentation of Code

The current code we used and created for implementation is provided with full documentation at the following URL: https://github.com/Cl4ryty/Neurodynamics-Group5

4 Results and Discussion

4.1 Performance Comparison between ODEs

We implemented a total of 22 ODEs in the training: 3 ODEs for *linear first order*, 2 for *nonlinear first order*, 7 for *linear second order*, 5 for *nonlinear second order*, 3 for *linear third order*, and 2 for *nonlinear third order* category.[1]

Among the obtained mean values, we found 2 values that are extremely high compared to the other values. These 2 outliers are removed from the data. Only 19 ODEs[2] were taken into account while

[1] Different types of invalid results, such as NaN, led us to exclude several ODEs in data collection.

[2] One value is from the category of *linear second order*, with a reported value of 39327.52 seconds and the other one is from *linear third order* with 2694.06 seconds. There is also a missing data reported with NaN value found under *linear third*

plotting final losses (Figure 1 & 2). For the graph of training time (Figure 3 & 4), no outliers or missing data were found. Thus, all 22 ODEs were taken into account while plotting.

Figure 1 and 3 (page 11 & 12) plot the final loss and the training time of individual ODEs. Figure 2 and 4 (page 11 & 12) display the statistical distribution of final losses and the training time in the six categories. The straight line indicates the median value and the colored dot indicates the mean in each category. Table 3 and 4 (page 13) show the numerical values from the Figure 2 and 4.[3]

According to our code implementation, the computation of the loss requires more derivations for higher order DEs, e.g. for first order DEs only the first derivative is calculated and for second order DEs the first and second derivatives are calculated. Therefore, computationally, the total training time should increase with the order of DEs. Furthermore, linear DEs are analytically easier to solve than nonlinear DEs (Huntley et al., 1983).[4] This reasoning led us to our first hypothesis – the performance of the ANN decreases as the complexity of the ODE increases, that is, when the order of the ODE increases and when it is a nonlinear ODE compared to when it is linear.

It is clear that the median final losses of *linear first order* and *linear third order* are less than *nonlinear first order* and *nonlinear third order* respectively. However, the mean of nonlinear ODEs are bigger than linear ODEs in the category of *third order*. It can also be observed that the median of *linear first order* and *nonlinear second order* are clearly lower than *linear second order* and *nonlinear third order* respectively. Nonetheless, the mean and the median of *linear second order* and *nonlinear first order* are greater than *linear third order* and *nonlinear second order* respectively, which is contradictory to our hypothesis.

It is shown that there is a correlation between the order of the ODEs and the training time, since the mean and median training time increase (or similar in case between *nonlinear first order* and *nonlinear second*) in the order of *first order*, *second order*, *third order* in each linear and nonlinear category.

When we compare between ODE categories with different linearity but the same order, however, the difference in mean and median is somewhat small. Most of the nonlinear categories have higher training time in mean and median, except the median and mean values of *second order* categories and the mean value of *third order* categories.

---

order category.

[3] Plots of final losses (Figure 1 & 2) are plotted in a logarithmic scale, whereas plots of training time (Figure 3 & 4) are plotted in a linear scale.

[4] General solutions exists in linear DEs, while in most of the cases, for nonlinear DEs general solutions do not exist and the solution for nonlinear DEs may be problem specific (Huntley et al., 1983).

4.2 Problems during the Development of the Software

Many of the problems encountered during development can be traced back to the lack of compatibility of modules and the severely incomplete documentation of software. As a result, the development took significantly more time than was planned in the schedule.

At the beginning, we considered two different possible approaches for obtaining the SNN from the ANN implemented with TensorFlow, one with the SNN toolbox, another with Nengo. In order to establish better comparability between both converted SNNs, we tried to find a common back-end on which both models could run well and which we would also be able to access, and came to the conclusion that SpiNNaker would be a good choice.

However, during the implementation we faced a lot of issues converting our ANN with Nengo due to using a custom loss function, which contains the DE that the model is trained to learn. It turned out that the converted Nengo ANN already delivered strongly deviating results from our original TensorFlow ANN such that we decided not to move forward with turning it into an SNN. After time-consuming testing, we came to the conclusion that there is no possibility to adapt the model in such a way that it delivers both comparable and correct results.

This changed our plan to using TensorFlow to train the ANN and then converting it to an SNN using the SNN toolbox. Unfortunately, the incomplete documentation of the SNN toolbox caused us major difficulties.[5] Also, only a limited selection of activation functions was usable with the SNN toolbox. In addition, we had to switch to older versions of libraries which are required by the SNN toolbox or modify the libraries' code to work with newer versions, which led to problems in interoperability when further working with the model. Implementing negative inputs on SNNs was not possible and inputs had to be rescaled to integer values to work with the SNN toolbox.

To potentially establish comparability between ANN and SNN, we used the free version of Google Colab cloud to train the ANN models. Yet, the ability to use a GPU runtime became limited after extended use. Also, the execution can be aborted due to "inactivity" and runtimes can only run for a certain (varying) time span until the notebook gets disconnected, which lead to several of our computations being aborted. Only after taking care of the execution by personally supervising it throughout the entire time the code was executed completely, and we obtained usable results. Unlike the expected, it is not

[5] Some core functionalities are not sufficiently explained and could only be understood by studying the code or by trial-and-error-methods, both of which were very time-consuming. In particular, the conversion of the trained model led to errors, with many problems resulting from the use of a custom loss function, which were only solvable by modifying the code of the SNN toolbox.

possible to measure the power consumption by using Colab.

The model converted with the SNN toolbox and generated by SpiNNaker gave an invalid result on SpiNNaker for some unknown reason. After extensive debugging, we could not solve the problem. It is also not possible to run the SNN toolbox on SpiNNaker, as this requires Tensorflow to be installed and the available memory on SpiNNaker was not sufficient. In general, there was little available information on how to deal with error messages due to a lack of users. This has resulted in us not being able to use SpiNNaker until now to convert ANNs to SNNs.

Problems in lack of compatibility of and documentation of Nengo and the SNN toolbox, as well as the technical specifications of SpiNNaker led to no or invalid results. This led to a delay in the schedule.

5 Conclusion

While analyzing the performance between ANNs solving ODEs of different categories, it was difficult to conclude that there is a correlation between the size of final loss and the complexity of ODEs, since there were several contradictory cases where the size of final loss increases as the complexity decreases. On the other hand, we had more compatible results with the training time. Most of the cases were consistent with our hypothesis where the size of training time increases as the complexity increases. However, we must mention that the sizes of the training time were similar when same order ODEs with different linearity are compared. The opposing result to our hypothesis might be explained by an insufficient amount of data. We conclude that further research is needed with a larger collection of DEs, which might lead to clearer results.

As for our aim to analyze performances between SNNs and ANNs, the SNN model did not provide usable results by the time of submission.

References

Cuomo, S., Di Cola, V. S., Giampaolo, F., Rozza, G., Raissi, M., & Piccialli, F. (2022). Scientific machine learning through physics-informed neural networks: Where we are and what's next. *arXiv preprint arXiv:2201.05624*.

Deng, L., Wu, Y., Hu, X., Liang, L., Ding, Y., Li, G., Zhao, G., Li, P., & Xie, Y. (2020). Rethinking the performance comparison between snns and anns. *Neural networks*, *121*, 294–307.

Han, J.-K., Yun, S.-Y., Lee, S.-W., Yu, J.-M., & Choi, Y.-K. (2022). A review of artificial spiking neuron devices for neural processing and sensing. *Advanced Functional Materials*, 2204102.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*, *4*(2), 251–257.

Huntley, I., Johnson, R., & Johnson, R. (1983). *Linear and nonlinear differential equations*. Ellis Horwood series.

Javanshir, A., Nguyen, T. T., Mahmud, M. A. P., & Kouzani, A. Z. (2022). Advancements in Algorithms and Neuromorphic Hardware for Spiking Neural Networks. *Neural Computation*, *34*(6), 1289–1328. https://doi.org/10.1162/neco_a_01499

Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., & Yang, L. (2021). Physics-informed machine learning. *Nature Reviews Physics*, *3*(6), 422–440.

Kharazmi, E., Zhang, Z., & Karniadakis, G. E. (2019). Variational physics-informed neural networks for solving partial differential equations. *arXiv preprint arXiv:1912.00873*.

Kundu, S., Datta, G., Pedram, M., & Beerel, P. (2021). Spike-thrift: Towards energy-efficient deep spiking neural networks by limiting spiking activity via attention-guided compression. https://doi.org/10.1109/WACV48630.2021.00400

Li, C., & Furber, S. (2021). Towards biologically-plausible neuron models and firing rates in high-performance deep spiking neural networks. *International Conference on Neuromorphic Systems 2021*. https://doi.org/10.1145/3477145.3477146

Paugam-Moisy, H., & Bohte, S. M. (2012). Computing with spiking neuron networks. *Handbook of natural computing*, *1*, 1–47.

Rueckauer, B., Hu, Y., Lungu, I., Pfeiffer, M., & Liu, S.-C. (2017). Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Front. Neurosci.* https://doi.org/10.3389/fnins.2017.00682

Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T., & Maida, A. (2019). Deep learning in spiking neural networks. *Neural Networks*, *111*, 47–63. https://doi.org/10.1016/j.neunet.2018.12.002

Appendix

Figure 1: Individual final losses.

Figure 2: Final losses according to the 6 categories.

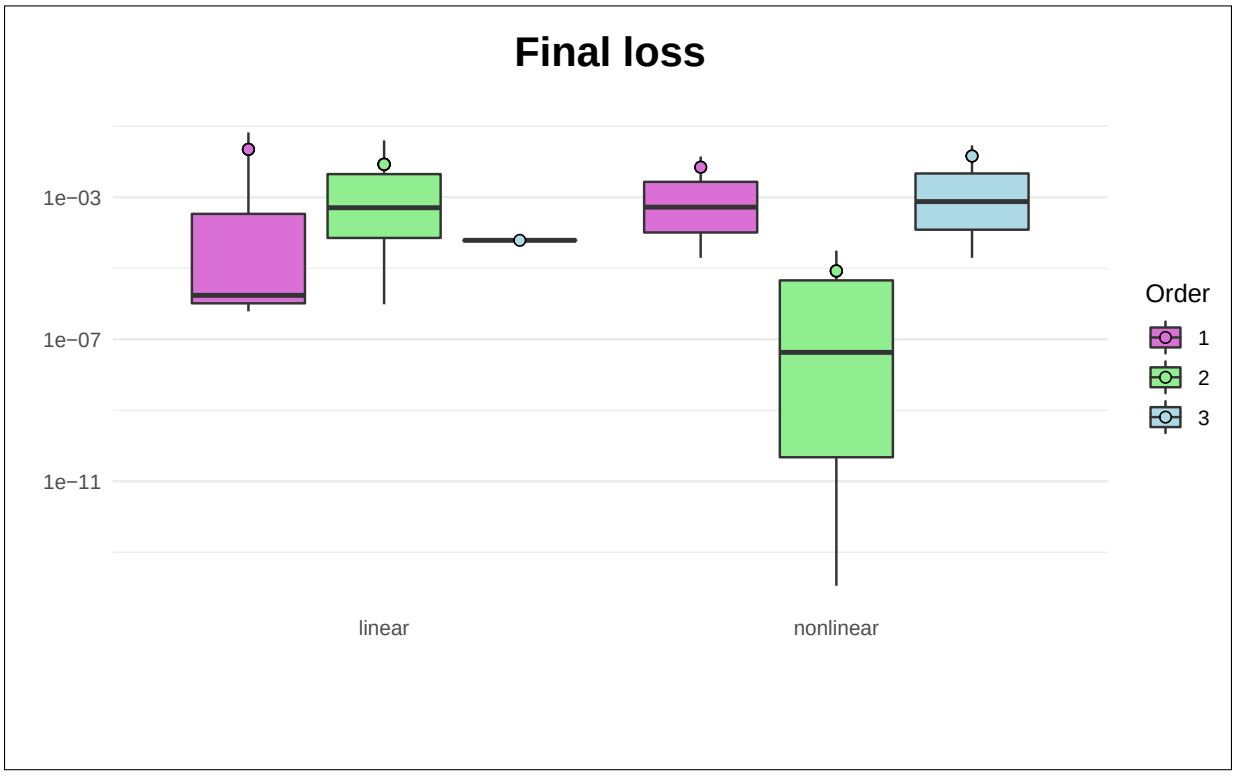

Figure 3: Individual training time.

Figure 4: Training time according to the 6 categories.

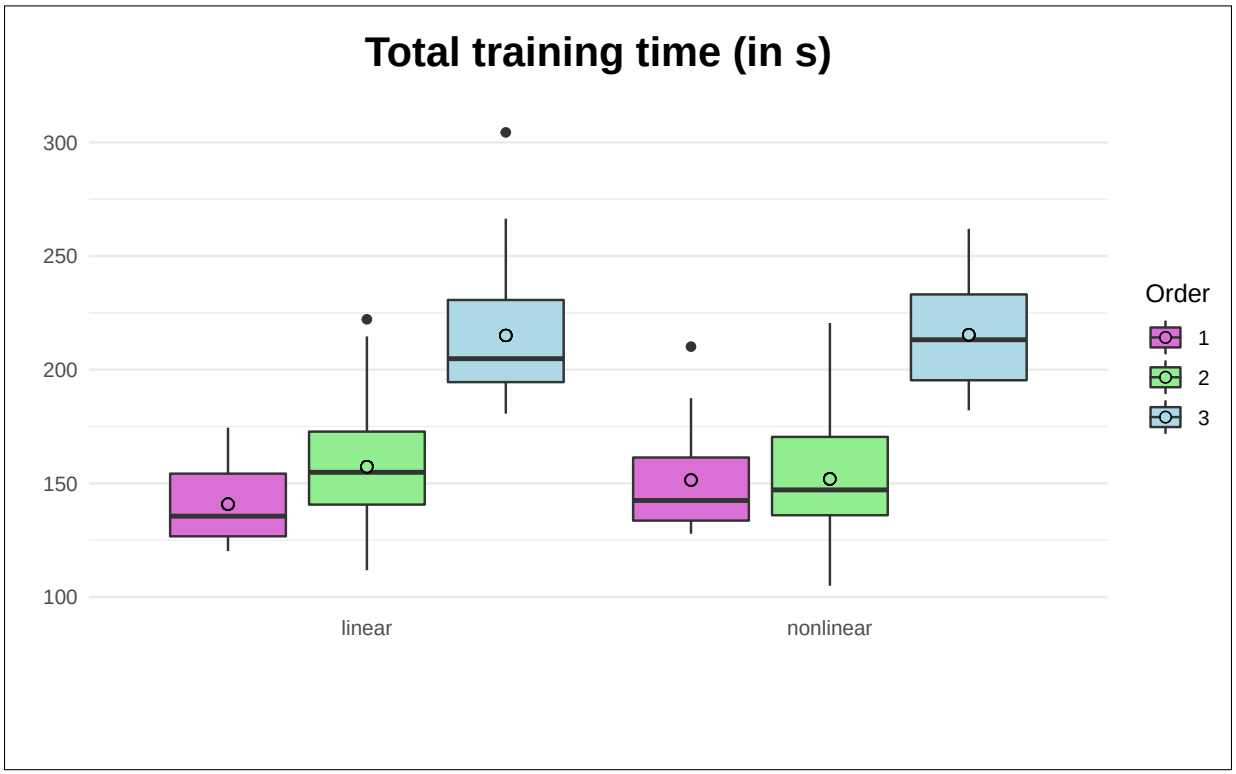

Table 3: Mean final losses and mean training time according to the 6 categories.

Category of DEs	**Mean Final Losses**	**Mean Training Time [s]**
Linear First Order	2.24e-02	140.86
Linear Second Order	8.47e-03	157.32
Linear Third Order	6.13e-05	215.05
Nonlinear First Order	7.05e-03	151.46
Nonlinear Second Order	8.43e-06	151.93
Nonlinear Third Order	1.45e-02	215.39

Table 4: Median final losses and median training time according to the 6 categories.

Category of DEs	**Median Final Losses**	**Median Training Time [s]**
Linear First Order	1.73e-06	135.57
Linear Second Order	5.08e-04	154.87
Linear Third Order	6.13e-05	204.84
Nonlinear First Order	7.05e-03	142.48
Nonlinear Second Order	1.20e-06	147.13
Nonlinear Third Order	1.45e-02	213.18